# Classes em Java

Programação II

2015/16

# Paradigma: Programação por Objetos (OOP/POO/PPO)

- Conceito de OBJETO
- abstracção que representa:
  - ➡ entidade única
  - ➡ estrutura
  - ➡ comportamento

# Paradigma POO

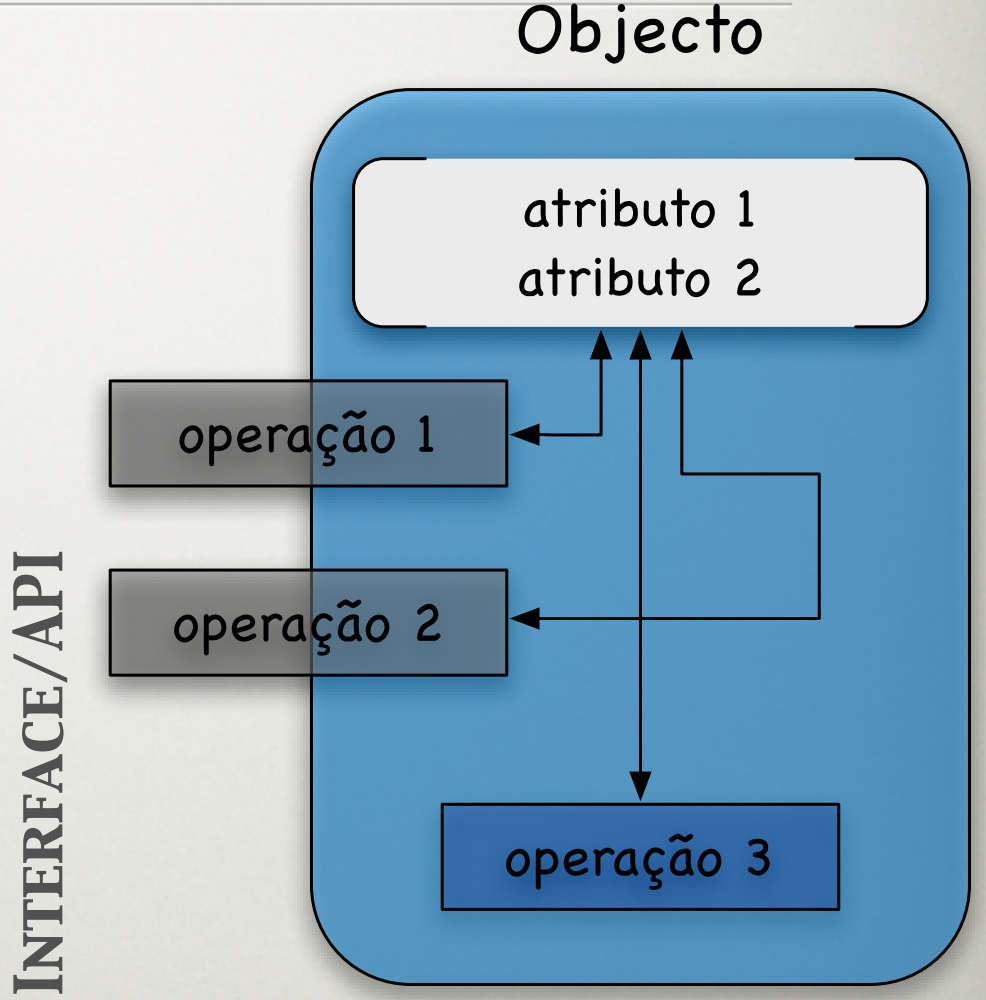
- Exemplo:
  - Ponto do Plano
    - Abstração
    - Atributos (Características)
    - Comportamento (operações que se podem realizar)

# Paradigma POO

- Ponto2D
  - Atributos
    - coordenada em X
    - coordenada em Y
  - Operações
    - obter ou modificar coordenada em X
    - obter ou modificar coordenada em Y

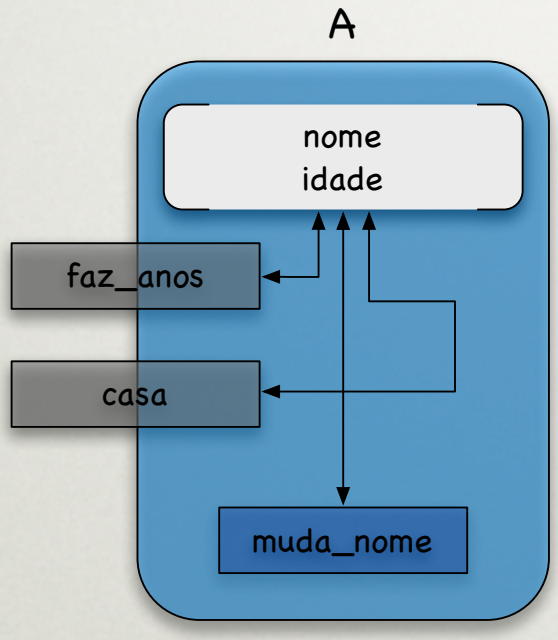
# Classes — Caixinha

- **PADRÃO para OBJETOS**
- Independente do contexto
- Reutilização de código
- Fiabilidade (erros)
- Modularidade

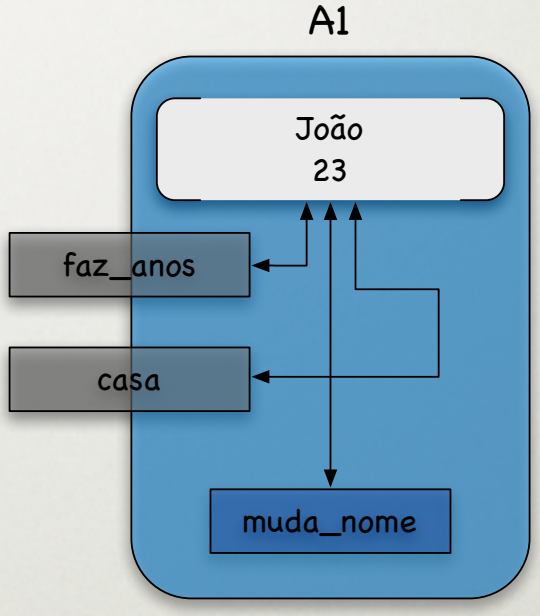


# Classes e Instâncias

(uma classe)

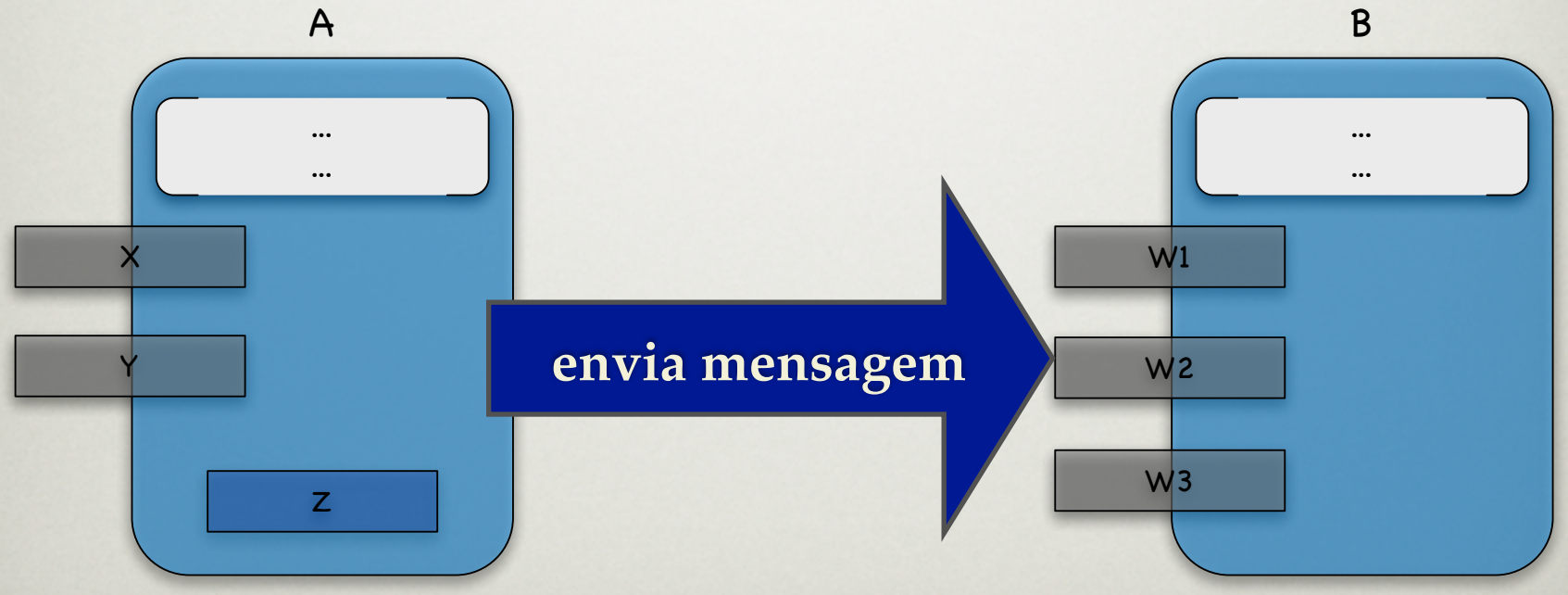



(uma instância)



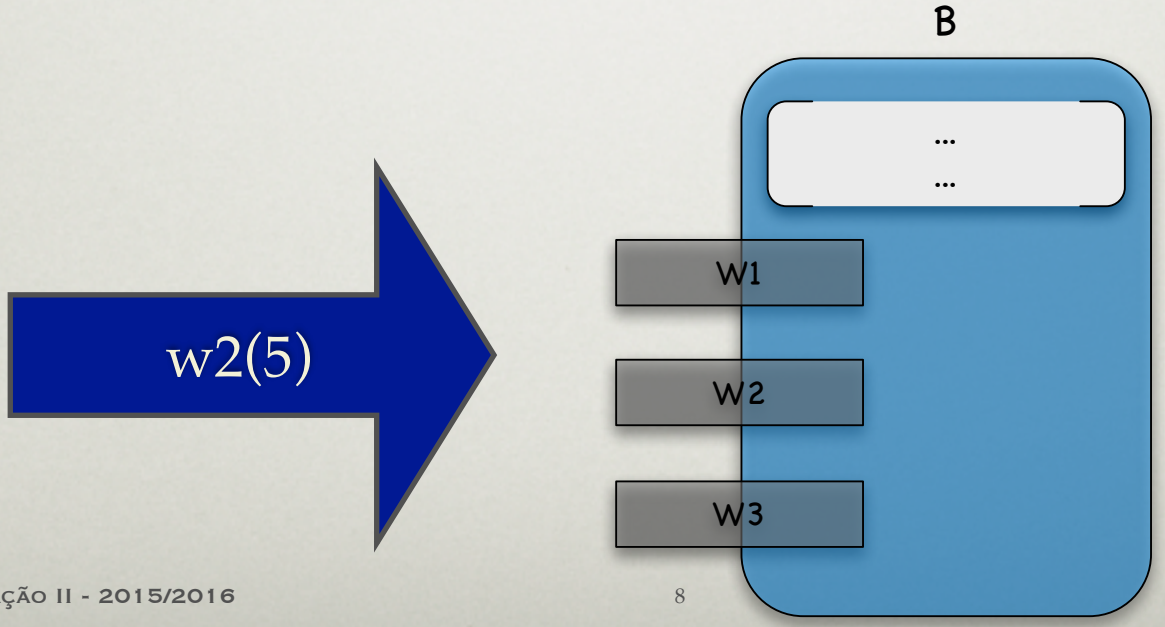
# Mensagens

- interação com um objeto feita pelo **envio duma mensagem**



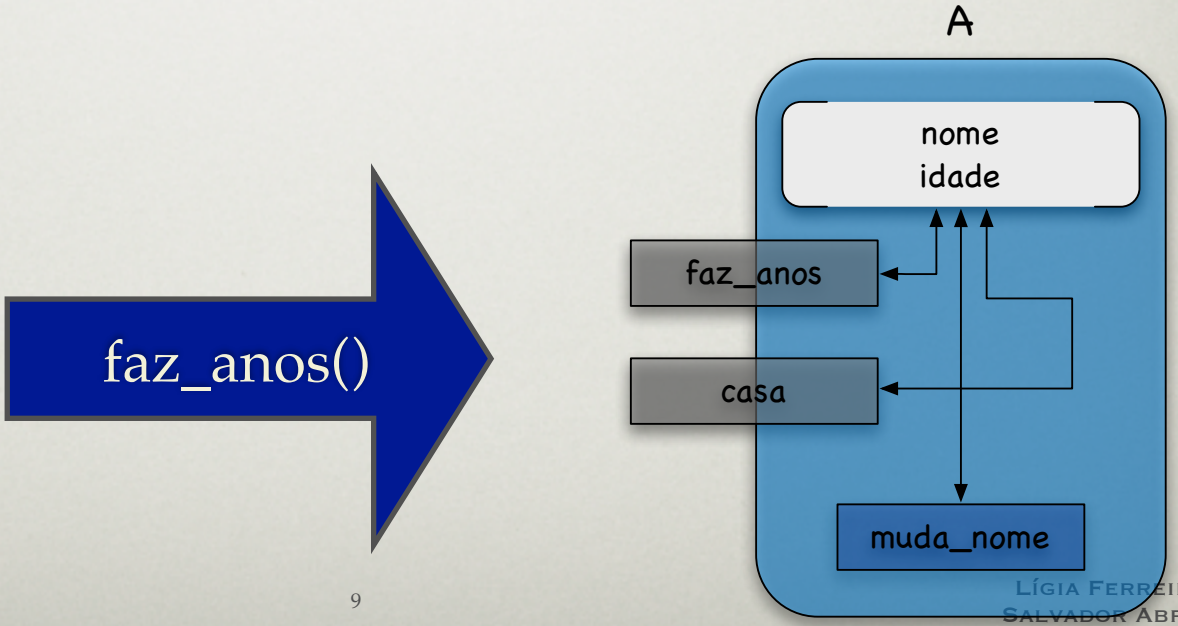
# Mensagens

- A **receção** duma **mensagem** ativa um **método**
- O método a executar é dado pelo **identificador** e **parâmetros** da messagem.



# Mensagens

- computação consiste na execução de métodos
- execução dum método pode
  - ➡ modificar o estado
  - ➡ enviar novas mensagens



# Mensagens

- recetor.mensagem();
  - p2.print();
- recetor.mensagem(arg1,arg2,...,argn);
  - p2.setX(3);
- valor=recetor.mensagem();
  - X=p2.getY();
- valor=recetor.mensagem(arg1,arg2,...,argn)

# Classes

- objetivo: garantir que objetos semelhantes têm um padrão
  - ➡ mesma estrutura
  - ➡ mesmo comportamento
- abordagens
  - ➡ Copy-Paste
  - ➡ outro objeto que represente
    - a estrutura deste objeto
    - a interface deste objeto
    - seja um **padrão** para objetos como este
- chama-se uma classe

# Classes

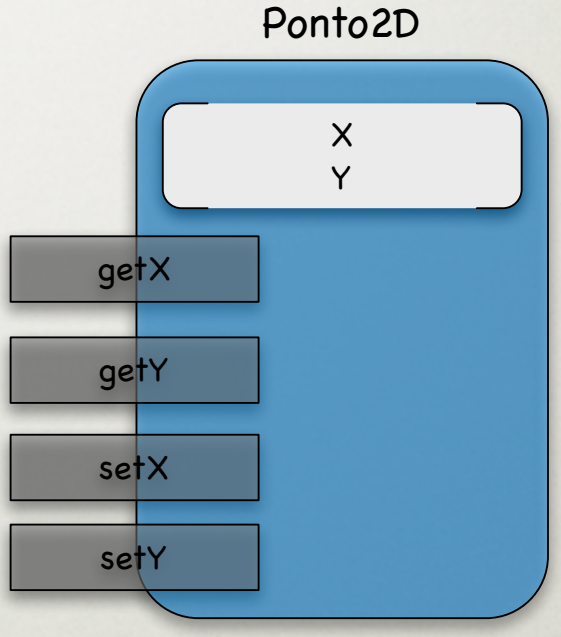
- uma classe
  - ➡ Define estrutura e comportamento dos objetos
  - ➡ Dá mecanismo para criar novos objetos dessa classe (instâncias)

# Classes

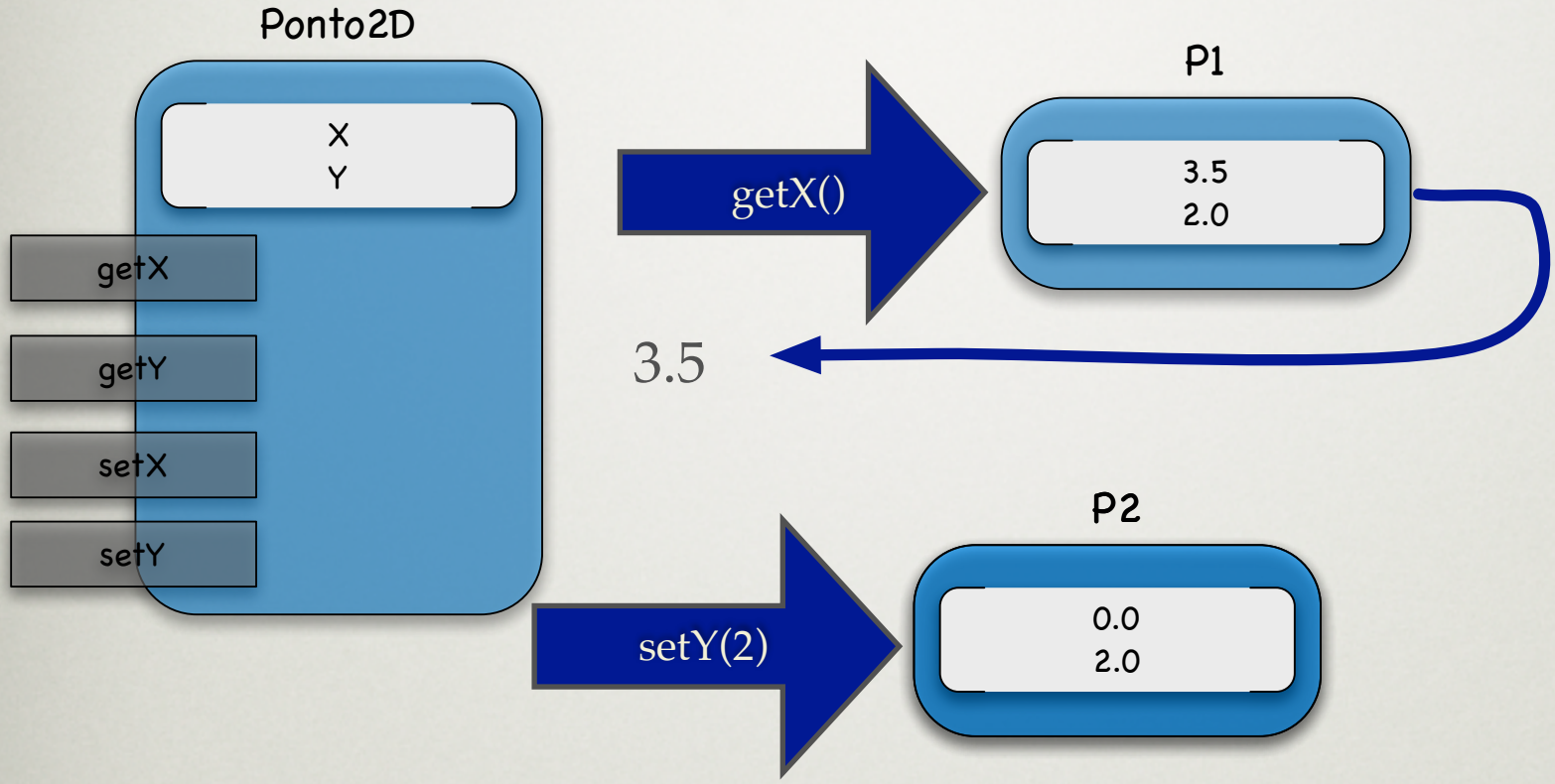
- Por exemplo, com a classe Ponto2D:

  ➡ Os pontos2D são caracterizados por:
  - coordenada em X
  - coordenada em Y

  ➡ Para Pontos2D concretos (instâncias), podemos:
  - Saber suas coordenadas
    - getX(), getY()
  - Alterar as coordenadas
    - setX(x), setY(y)



# Classes e Instâncias



# Estrutura

- São as **variáveis de instância**
  - colocado logo após o nome da classe
  - parecem variáveis locais:
    - id_tipo id_var[=valor] [,id_var[=valor]]... ;
  - Exemplo:
    - float coordX=0, coordY=1.25f;
  - o tipo pode ser básico ou composto

# Variáveis de instância

- acesso dentro da classe
  - usa-se o nome (identificador)
  - como uma variável normal
- acesso fora da classe
  - nome **qualificado**
    - INSTANCIA.VARIAVEL
  - ex
    - ponto1.coordX
  - acesso direto a variáveis de **outras instâncias** é **proibido**
    - exceto no caso de variáveis **public**

# Comportamento

- Comportamento é programado por **métodos de instância**
- sintaxe: cabeçalho(header) e corpo(body)
- um cabeçalho
  - ➡ <tipo_do_resultado> identificador ( tipo1 parametro1, ...)
    - É obrigatória a especificação do tipo do retorno
    - Caso não interesse há o tipo "vacuoso"-> void

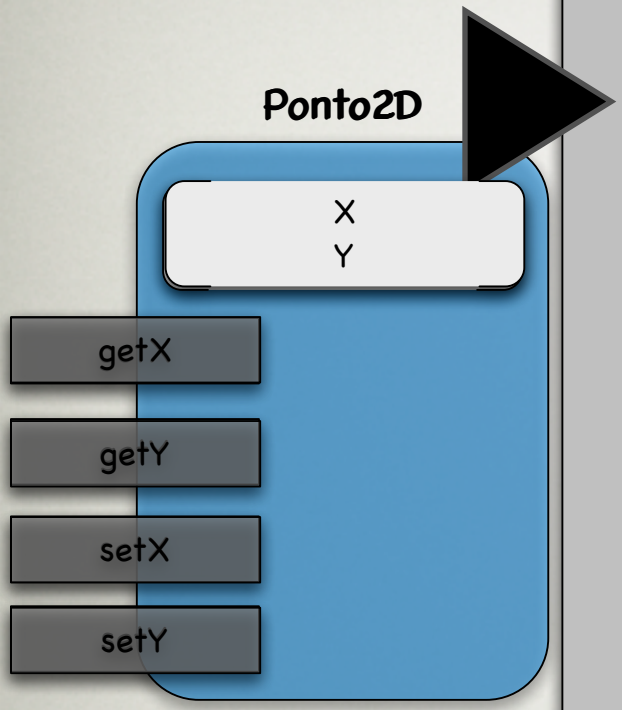
# Métodos de Instância

- Exemplos de cabeçalhos
  - ➡ double getX ()
  - ➡ void setX (double x)
- A **assinatura** dum método é dada pelo nome do método e pelos parâmetros, em número, tipo e ordem
- A assinatura identifica univocamente os métodos duma classe

# Métodos de Instância

- O corpo é um bloco de instruções entre chavetas
- Exemplos
  - ➡ { return coordX; }
  - ➡ { coordX = x; }
  - ➡ Quando o retorno não é void é obrigatório terminar com uma instrução **return**:
    - return exp;
      - exp é uma expressão do **mesmo tipo** do valor a retornar
      - após o return não é executada mais nenhuma instrução

# Ponto2D.java




```
class Ponto2D {

    double x;
    double y;

    double getX() {
        return x;
    }

    double getY() {
        return y;
    }

    void setX (double a) {
        x=a;
    }

    void setY (double a) {
        y=a;
    }
}
```

# Classes

- para criar instancias
- introduz o operador **new**, que antecede o nome da classe e mais parâmetros.
- o método invocado tem exactamente o nome da classe.
  - ➡ `new Ponto2D();`
  - ➡ `Ponto2D p1=new Ponto2D();`
- chama-se um métodos **construtor**

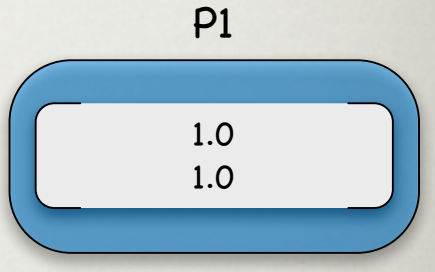
# Métodos construtores

- criam instâncias
- são métodos, logo obedecem às regras de definição de métodos, com algumas particularidades:
  - o identificador é o nome da classe
  - não tem valor de retorno
  - se não for fornecido nenhum na definição da classe, o Java fornece um por omissão (sem parâmetros)

# Métodos construtores

- O construtor por omissão:

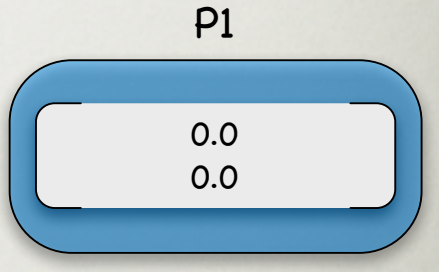
```
class Ponto2D{
    float coordX=1;
    float coordY=1;
    public static void main (String[] args) {
     Ponto2D p1=new Ponto2D();
    }
}
```



# Métodos construtores

- O construtor por omissão:

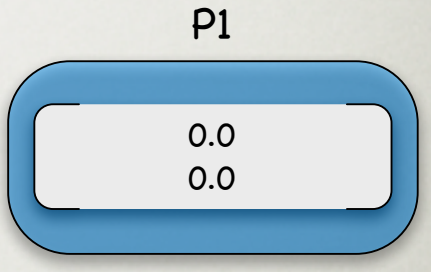
```
class Ponto2D{
        float coordX;
        float coordY;
        public static void main(String[] args){
                Ponto2D p1=new Ponto2D();
        }
}
```



# Métodos construtores

- O construtor por omissão pode ser redefinido:

```
class Ponto2D{
    float coordX=1;
    float coordY=1;
    Ponto2D(){
        coordX=0;
        coordy=0;
    }
    public static void main(String[] args){
        Ponto2D p1=new Ponto2D();
    }
}
```

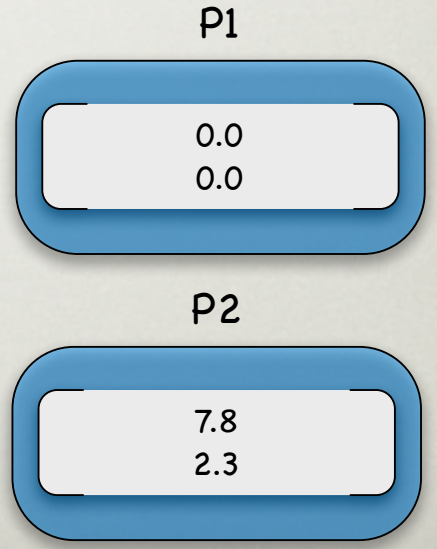


# Métodos construtores

- Outros construtores podem ser definidos

```
class Ponto2D{
        float coordX=1;
        float coordY=1;
        Ponto2D(){
            coordX=0;
            coordY=0;
        }
        Ponto2D(float x,float y){
            coordX=x;
            coordY=y;
        }
}
```

Que mensagens e a quem devem ser enviadas para obter estes objectos?



# Métodos construtores

- Quando se define um construtor o "default" deixa de existir

```
class Ponto2D{
      float coordX=1;
      float coordY=1;
      Ponto2D(float x,float y){
            coordX=x;
            coordY=y; }
      public static void main(String[] args) {
            Ponto2D p1=new Ponto2D(); }
}
```